

6800 Hardware - Introduction

The hardware section of this manual gives a detailed description of the various elements contained within the 6800 family of integrated circuits. Although there are many other components on each of the boards within the SWTPC 6800 Computer System, their purpose is to provide the clocks, buffering and address decoding for the 6800 parts described in this section. In order to be able to program the system, some knowledge of the hardware involved, especially the architecture of the microprocessor chip, is absolutely essential. One must have a thorough understanding of the operation and use of the various internal registers and the system's response to the various interrupts. Familiarity with both the binary and hexadecimal number system is absolutely essential.

Pages MPU-1 thru MPU-19 describe the architecture as well as the various data and control lines of the 6800 microprocessor element. The descriptions given for the various data and control lines on pages MPU-8 thru MPU-11 need not be read since the material may be confusing and is not related to programming the system.

Pages MPU-20 thru MPU-30 describe the relations and conversion between decimal (base 10), binary (base 2), octal (base 8) and hexadecimal (base 16) number systems. This section is extremely important and if not understood must be supplemented with additional material. The programmer must have a firm background in especially the binary and hexadecimal number systems before any programming can be done. Even the mini-operating system described in Engineering Note 100 within this notebook looks for all address and data entry in hexadecimal notation. An understanding of the binary representation is important since all data contained within the system's internal registers is stored in binary form.

Pages PIA-1 thru PIA-14 discuss the data and control lines on, and registers within the 6820 peripheral interface adaptor integrated circuit used on the MP-C serial, control interface and MP-L parallel interface boards. It is not necessary to be familiar with this section to use the MP-C serial, control interface board supplied with the MP-68 system, however, it is recommended reading when using the MP-L parallel interface option.

Pages ACIA-1 thru ACIA-9 discuss the data and control lines on, and registers within the 6850 asynchronous communications adaptor integrated circuit used on the MP-S Serial interface board. This section is recommended reading only when using the MP-S serial interface option.

The "Motorola M6800 Systems Reference and Data Sheets" booklet and a data sheet on the MC14411 bit rate generator have been included as additional information on the major elements within SWTPC 6800 Computer System. Much of the material contained within the booklet has already been talked about in the preceeding sections, however, it is suggested reading which may help clear up any misunderstandings you may have about elements within the system.

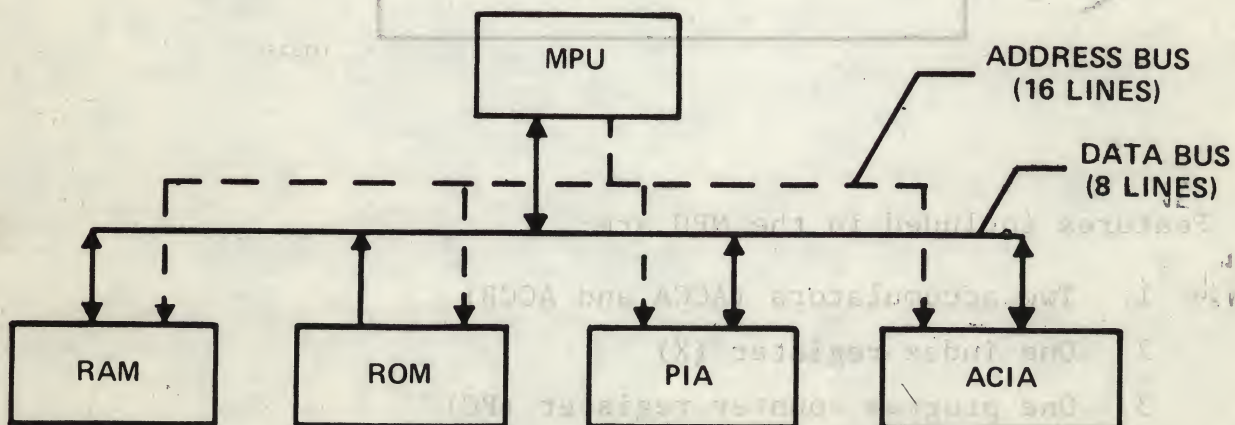
Chapter 3 of the "Motorola 6800 Programming Manual" is also recommended as supplemental material before proceeding to the Programming section of this notebook.

INTRODUCTION

The Motorola M6800 Microcomputer System of standard LSI (Large Scale Integration) devices permits the systems designer to configure and connect a total system with a minimum amount of time and effort. The MC6800 Microprocessing Unit (MPU) forms the nucleus of the system. LSI modules available which may be used to configure a total system in conjunction with the MC6800 MPU, include: 1) MC6810 Random Access Memory (RAM); 2) MC6830 Read Only Memory (ROM); 3) MC6820 Peripheral Interface Adapter (PIA), and 4) MC6850 Asynchronous Communications Interface Adapter (ACIA).

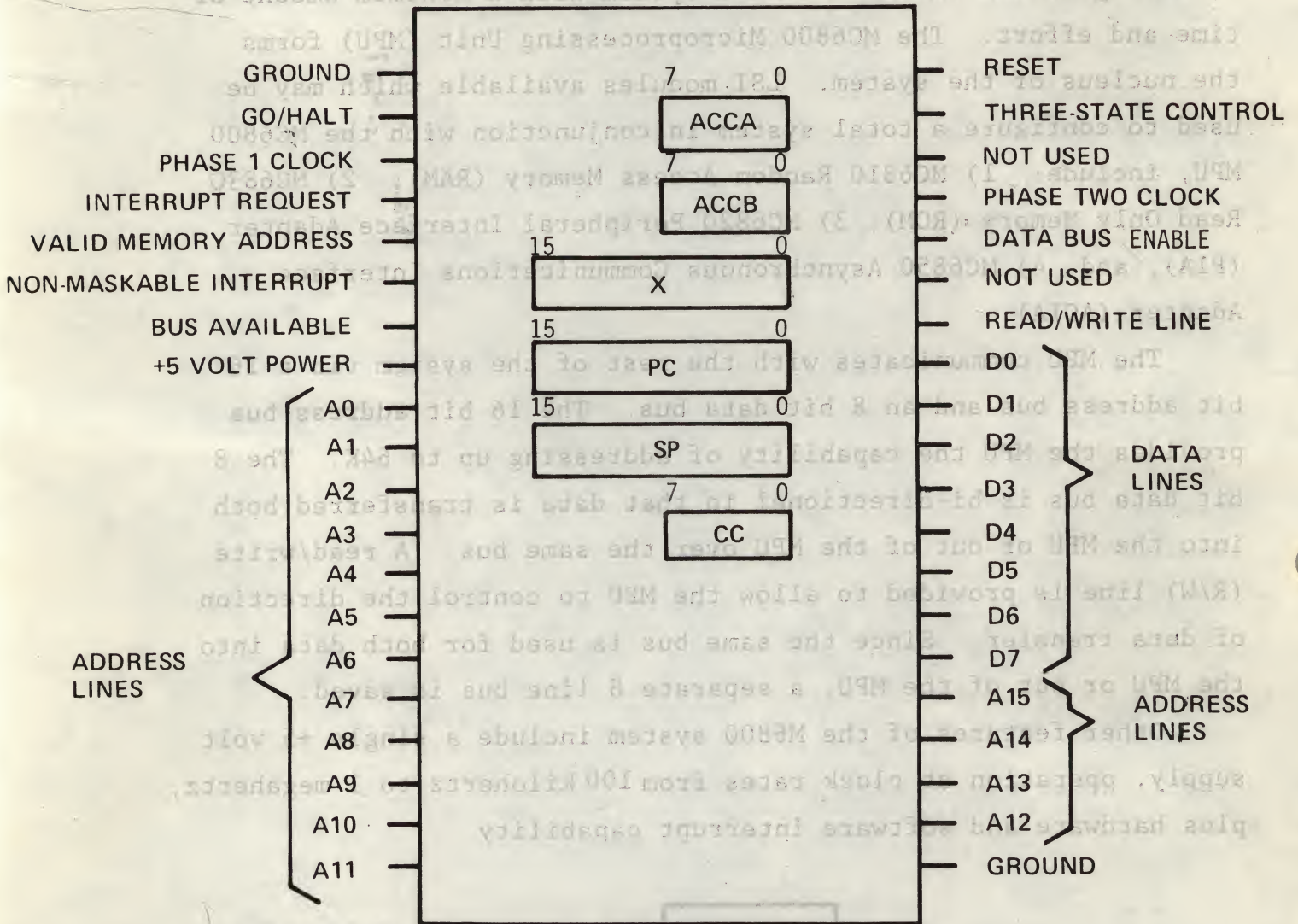
The MPU communicates with the rest of the system via a 16 bit address bus and an 8 bit data bus. The 16 bit address bus provides the MPU the capability of addressing up to 64K. The 8 bit data bus is bi-directional in that data is transferred both into the MPU or out of the MPU over the same bus. A read/write (R/W) line is provided to allow the MPU to control the direction of data transfer. Since the same bus is used for both data into the MPU or out of the MPU, a separate 8 line bus is saved.

Other features of the M6800 system include a single +5 volt supply, operation at clock rates from 100 kilohertz to 1 megahertz, plus hardware and software interrupt capability.



Microprocessing Unit (MC6800)

The nucleus of the M6800 Microcomputer Family is the microprocessing unit (MPU). The MPU is enclosed in a 40 pin package as shown below:



D1527

Features included in the MPU are:

1. Two accumulators (ACCA and ACCB)
2. One index register (X)
3. One program counter register (PC)
4. One stack pointer register (SP)

5. One condition code register (CC)
6. 72 instructions
7. Five addressing modes
8. System clock range of 100 kilohertz to 1 megahertz
9. Program interrupt capability

Accumulators

The MPU contains 2 accumulators designated ACCA and ACCB. Each accumulator is 8 bits (one byte) long and is used to hold operands and data from the arithmetic logic unit. Instructions which involve one or both accumulators are:

ABA - Add accumulator A to accumulator B
ADC - Add with carry
ADD - Add without carry
AND - Logical AND
ASL - Arithmetic shift left
ASR - Arithmetic shift right
BIT - Bit test
CBA - Compare accumulators
CLR - Clear
CMP - Compare
COM - Complement
DAA - Decimal adjust ACCA
DEC - Decrement
EOR - Exclusive OR

INC - Increment
 LDA - Load accumulator
 LSR - Logical shift right
 NEA - Negate
 ORA - Inclusive OR
 PSH - Push data onto stack
 PUL - Pull data from stack
 ROL - Rotate left
 ROR - Rotate right
 RTI - Return from interrupt
 SBA - Subtract accumulators
 SBC - Subtract with carry
 STA - Store accumulator
 SUB - Subtract
 SWI - Software interrupt
 TAB - Transfer from accumulator A to accumulator B
 TAP - Transfer from accumulator A to processor condition
 codes register
 TBA - Transfer from accumulator B to accumulator A
 TPA - Transfer from processor condition codes register to
 accumulator A
 TST - Test
 WAI - Wait for interrupt

Index Register

The index register (X) is a 16 bit (2 byte) register which is primarily used to store a memory address in the Indexed mode of memory addressing. The index register may be decremented, incremented and stored. Instructions which involve the index register are:

CPX - Compare index register
 DEX - Decrement index register
 INX - Increment index register
 LDX - Load index register
 RTI - Return from interrupt
 STX - Store index register
 SWI - Software interrupt
 TSX - Transfer stack pointer to index register
 TXS - Transfer index register to stack pointer
 WAI - Wait for interrupt

Program Counter

The program counter (PC) is a 16 bit register that contains the address of the next byte to be fetched from memory. When the current value of the program counter is placed on the address buss, the program counter will be incremented automatically.

Stack Pointer

The Stack Pointer (SP) is a 16 bit (? byte) register that contains a beginning address, normally in RAM, where the status of the MPU registers may be stored when the MPU has other functions to perform, such as during an interrupt or during a Branch to Subroutine (BTS). The address in the stack pointer is the starting address of sequential memory locations in RAM where MPU status registers will be stored. The status of the MPU will be stored in the RAM as follows:

Stack Pointer Address	:	contents of PCL
Stack Pointer Address-1	:	contents of PCH
Stack Pointer Address-2	:	contents of IXL
Stack Pointer Address-3	:	contents of IXH
Stack Pointer Address-4	:	contents of ACCA

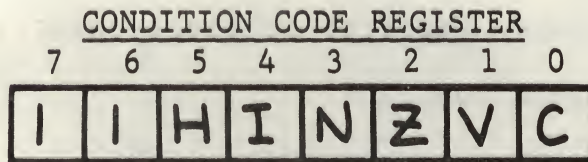
Stack Pointer Address-5 : Contents of ACCB

Stack Pointer Address-6 : Contents of CC

After the status of each register is stored on the stack, the Stack Pointer will be decremented. When the stack is unloaded (status of registers restored), the status of the last register on the stack will be the first register that is restored.

Condition Code Register (CC)

The condition code register is an 8 bit register. Each individual bit may get set or get cleared from execution of an instruction. To see how each instruction effects the condition code register, refer to the M6800 programming manual. The primary use of this register is execution of the conditional branch instruction. Bit 6 and 7 are not used and remain at logic "1."



<u>BIT NO.</u>	<u>FUNCTION</u>
0	C (Carry-Borrow Test)
1	V (Overflow Test)
2	Z (Zero Test)
3	N (Negative Test)
4	I (Interrupt Mask Test)
5	H (Half Carry Test)

Carry-Borrow: For addition, the carry-borrow condition code (C) in the zero bit position, represents a carry. This bit gets set (C=1) to indicate a carry, and is reset (C=0) if there is no carry.

For subtraction, the C bit is set (C=1) to indicate a borrow and is reset (C=0) to indicate there was no borrow.

Overflow: The V bit (bit 1) of the condition code register is set (V=1) when two's complement overflow results from an arithmetic operation, and is reset (V=0) if two's complement overflow does not occur.

Zero: The Z bit (bit 2) of the condition code register is set (Z=1) if the result of an arithmetic operation is zero, and is reset (Z=0) if the result is not zero.

Negative: The N bit (bit 3) of the condition code register is set (N=1) if bit 7 of an arithmetic operation is set (equal to 1). This indicates that the two's complement number, represented by the bit pattern of the result, is negative. The N bit is reset (N=0) if bit 7 of the arithmetic result is equal to 0.

Interrupt Mask: If this I bit (bit 4) is set (I=1), the MPU cannot respond to an interrupt request from any peripheral device.

Half-Carry: The half carry bit H (bit 5) of the condition code register is set (H=1) during execution of any of the instructions ABA, ADC, or ADD, if there is a carry from bit position 3 to bit position 4. The half carry is reset (H=0) during these operations, if there is no carry from bit position 4.

MPU Signal Descriptions

1. READ/WRITE (R/W): This output line is used to signal all devices external to the MPU that the MPU is in a read state (R/W=High) or a write state (R/W=Low). The normal standby state of this line when no external devices are being accessed is a high state. This line is three-state. When three-state goes high, this line enters the high impedance mode.
2. VALID MEMORY ADDRESS (VMA): This output line, (when in the high state) tells all devices external to the MPU that there is a valid address in the address bus. For RAM's and ROM's, this line should be ANDed with $\phi 2$ clock and used as one of the enables. For PIA's, this line should be ANDed with one of the PIA address lines. This signal is not three-state.
3. DATA BUS ENABLE(DBE): This signal will enable the data bus drives when in the high state. This input is normally the phase 2 ($\phi 2$) clock. During the high state, it will permit data to be output during a write cycle. During an MPU read cycle, the data bus drives will be disabled internally.
4. INTERRUPT REQUEST(IRQ): This input from the PIA's requests that an interrupt sequence be generated within the machine. The processor will wait until it completes the current instruction that is being executed before it recognizes the request. At that time, if the interrupt mask bit in the Condition Code Register is not set (interrupt masked), the machine will begin an interrupt sequence. The Index Register, Program Counter, Accumulators, and Condition Code Register are stored away on the stack. Next the MPU will respond to the interrupt request by setting the interrupt mask bit high so that no further interrupts may occur. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations n-6 and n-7 where n is the highest ROM address. An address loaded at these locations causes the MPU to branch to an interrupt routine in memory.

5. Phase One (Ø 1)& Phase (Ø2)Clocks: These two pins are used for a two phase non-overlapping clock that runs at the V_{DD} voltage level. These clocks run at a rate up to 1 megahertz.
6. Restart (RES): RESTART (RES)--This input is used to start the MPU from a power down condition, resulting from a power failure or an initial start-up of the processor. If a positive edge is detected on the input, this will signal the MPU to begin the restart sequence. This will restart the MPU and start execution of a routine to initialize the processor. All the higher order address lines will be forced high. For the restart, the last 2 memory locations in the last ROM ($n \& n-1$) will be accessed, whereby an address is stored which is the address to be loaded in the program counter which tells the processor where program execution is to begin.
7. NON-MASKABLE INTERRUPT(NMI): This input requests that a non-mask-interrupt sequence be generated within the processor. As with the Interrupt Request signal, the processor will complete the current instruction that is being executed before it recognizes the NMI signal. The interrupt mask bit in the Condition Code Register has no effect on NMI. The Index Register, Program Counter, Accumulators, and Condition Code Register are stored away on the stack. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations $n-2$ and $n-3$. An address loaded at these locations causes the MPU to branch to a non-maskable interrupt routine in memory.
8. Go/Halt(G/H): When this input is in the high state, the machine will fetch the instruction

addressed by the program counter and start execution. When low all activity in the machine will be halted. This input is level sensitive. In the halt mode, the machine will stop at the end of an instruction. Bus Available will be at a logic "1" level. Valid Memory Address will be at a logic "0" and all other three-state lines will be in the three-state mode.

The halt line must go low with the leading edge of phase one to insure single instruction operation. If the halt line does not go low with the leading edge of phase one, one or two instruction operations may result, depending on when the halt line goes low relative to the phasing of the clock.

9. BUS AVAILABLE (BA): The Bus Available signal will normally be in the low state. When activated, it will go to the high state indicating that the MPU has stopped and that the address bus is available. This will occur if the GO/HALT line is in the Halt (low) mode or the MPU is in a "Wait" state as the result of some instruction, such as the WAI instruction.
10. THREE-STATE CONTROL (TSC): This input causes all of the address lines and the Read/Write line to go into the off or high impedance state. The Valid Memory address and Bus Available signals will be forced low. The data bus is not affected by TSC and has its own enable (Data Bus Enable). In DMA applications, the Three-State Control line should be brought high on the leading edge of the Phase One Clock. The $\phi 1$ clock must be held in the high state for this function to operate properly. The address bus will then be available for other devices to directly address memory. Since the MPU is a dynamic device, it must be refreshed periodically or destruction of data will occur.
11. ADDRESS BUS (A0/A15): Sixteen pins are used for the address bus. The outputs are three-state bus drivers capable of driving one standard TTL load and 130pf at 1 Megahertz.

When the output is turned off, it is essentially an open circuit. This permits the MPU to be used in DMA applications.

12. DATA BUS (DO/D7):

Eight pins are used for the data bus. It is bi-directional, transferring data to and from the memory and peripheral devices. It also has three-state output buffers capable of driving one standard TTL load and 130pf at 1 Megahertz.

Microprocessor Instruction Set--Alphabetic Sequence

ABA	Add Accumulators	INS	Increment Stack Pointer
ADC	Add with Carry	INX	Increment Index Register
ADD	Add		
AND	Logical And	JMP	Jump
ASL	Arithmetic Shift Left	JSR	Jump to Subroutine
ASR	Arithmetic Shift Right	LDA	Load Accumulator
		LDS	Load Stack Pointer
BCC	Branch if Carry Clear	LDX	Load Index Register
BCS	Branch if Carry Set	LSR	Logical Shift Right
BEQ	Branch if Equal to Zero		
BGE	Branch if Greater or Equal Zero		
		NEG	Negate
BGT	Branch if Greater than Zero	NOP	No Operation
BHI	Branch if Higher		
BIT	Bit Test	ORA	Inclusive OR Accumulator
BLE	Branch if Less or Equal	PSH	Push Data
BLS	Branch if Lower of Same	PUL	Pull Data
BLT	Branch if Less than Zero	ROL	Rotate Left
BMI	Branch if Minus	ROR	Rotate Right
BNE	Branch if Not Equal to Zero	RTI	Return from Interrupt
BPL	Branch if Plus	RTS	Return from Subroutine
BRA	Branch Always		
BSR	Branch to Subroutine	SBA	Subtract Accumulators
BVC	Branch if Overflow Clear	SBC	Subtract with Carry
BVS	Branch if Overflow Set	SEC	Set Carry
		SEI	Set Interrupt Mask
CBA	Compare Accumulators	SEV	Set Overflow
CLC	Clear Carry	STA	Store Accumulator
CLI	Clear Interrupt Mask	STS	Store Stack Register
CLR	Clear	STX	Store Index Register
CLV	Clear Overflow	SUB	Subtract
CMP	Compare	SWI	Software Interrupt
COM	Complement		
CPX	Compare Index Register	TAB	Transfer Accumulators
		TAP	Transfer Accumulators to Condition Code Reg.
DAA	Decimal Adjust	TBA	Transfer Accumulators
DEC	Decrement	TPA	Transfer Condition Code Reg. to Accumulator
DES	Decrement Stack Pointer	TST	Test
DEX	Decrement Index Register	TSX	Transfer Stack Pointer to Index Register
EOR	Exclusive OR	TXS	Transfer Index Register to Stack Pointer
INC	Increment	WAI	Wait for Interrupt

Hardware Interrupts

What happens when the MPU gets a hardware interrupt? After it has been determined that the interrupt is not non-maskable, the MPU checks the status of the mask bit (bit 4 of the condition code register). If the mask bit is set, the main program continues until a CLI (clears bit 4 of condition code register) instruction is executed, after which time the MPU will honor an interrupt by going to the stack pointer (SP) register and fetch an address which will be the 1st address in RAM where the status of the MPU registers will be stored during servicing of the interrupt.

SP	:	contents of program counter low
SP-1	:	contents of program counter high
SP-2	:	contents of index register low
SP-3	:	contents of index register high
SP-4	:	contents of accumulator A
SP-5	:	contents of accumulator B
SP-6	:	contents of condition code register

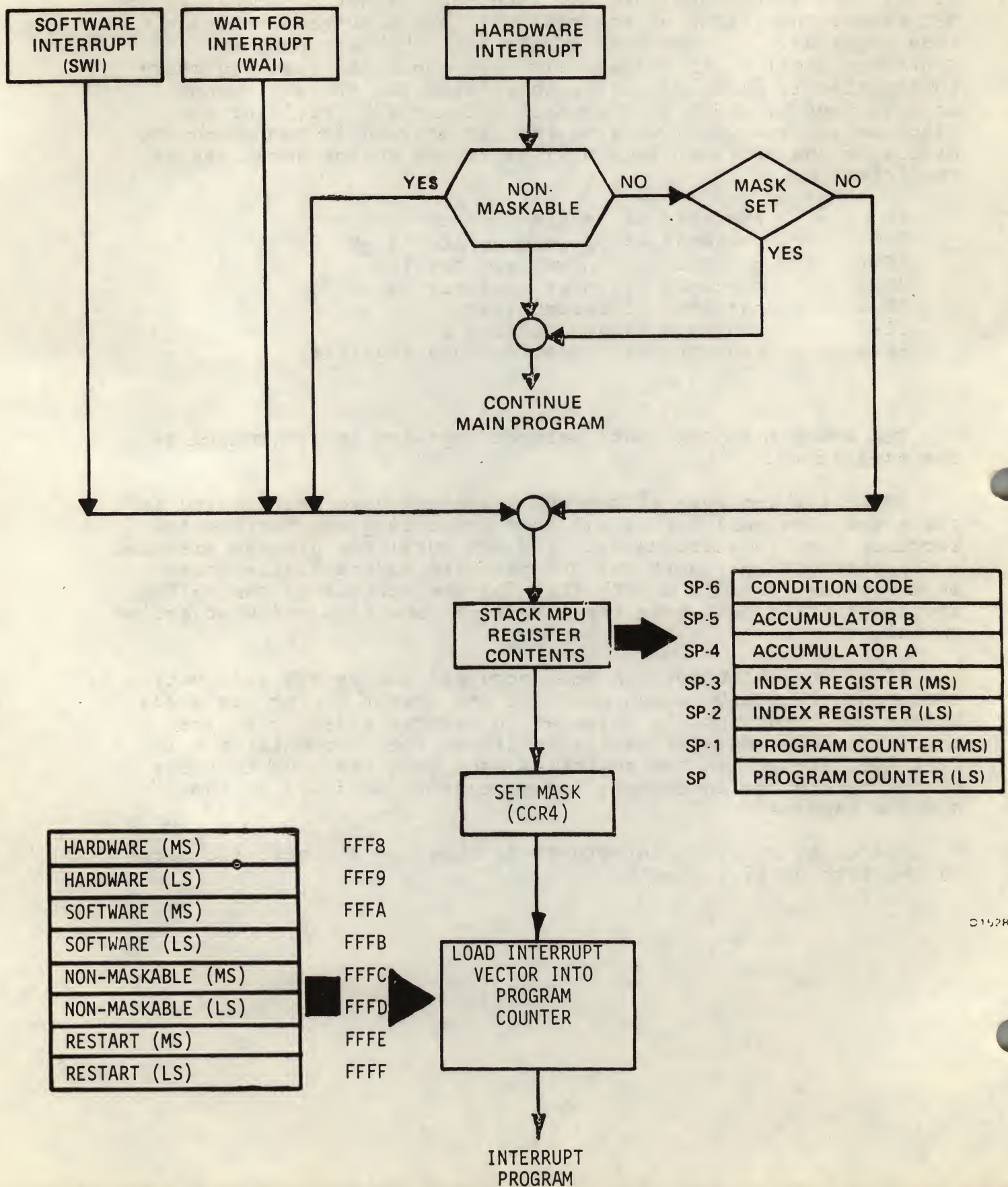
The address in the stack pointer register is determined by the programmer.

After the contents of the MPU registers have been stored in the stack, the mask bit is set thus preventing any further interrupts from interfering with the MPU until the program executes a CLI instruction. Next the MPU hardware automatically looks at addresses FFF8(MS) & FFF9 (LS) for the address of the polling routine to find out where the interrupt came from and what action to take.

After the interrupt has been serviced and an RTI instruction is executed, the stack, which contains the status of the registers before the interrupt, is unloaded in reverse order, i.e. the condition code register is loaded first, then accumulator B is restored, etc. When the registers have been restored to their status before the interrupt, the processor continues as though nothing happened.

The total story of interrupts is shown on the next two pages in the form of flow charts.

INTERRUPT FLOW CHART



SUMMARY OF MPU OPERATION

The MPU requires a two phase symmetrical, TTL compatible, non-overlapping clock. During the first phase of the clock (ϕ_1 high) an address will be placed on the address bus by the MPU. During the second phase of the clock (ϕ_2 high), the bidirectional data bus will be active. The first byte of an instruction enters the MPU and is transferred into an internal instruction register and decoded by the MPU. The MPU will then contain the information needed to read in an additional one or two bytes of program is necessary. Once the entire instruction is read into the MPU (one, two or three bytes) the instruction is then executed. The MPU then reads in the next sequential byte in the program and places it again in the instruction register. The program will sequentially be executed in this manner unless a branch or jump instruction changes the value of the program counter. If this occurs, the next instruction to be executed is determined by the new program counter value.

If an interrupt or reset occurs during this process, the program counter value will also be changed. The new program counter value is determined by the highest eight memory locations that are reserved for reset and interrupt vectors.

In the case of interrupt, the stack pointer is used to store the contents of the internal registers necessary to return to the program location prior to the interrupt. This happens when the interrupt program exits by an RTI (Return from interrupt instruction). Similarly, the stack pointer is used to store the program counter value when a JSR (Jump to Subroutine) or BSR (Branch to Subroutine) instruction occurs. The program counter returns to its original value when an RTS (Return

SUMMARY OF MPU OPERATION (Continued)

from Subroutine) instruction occurs. The stack pointer value is set by an LDS (Load Stack Pointer) instruction.

RESET SEQUENCE

1. While HALT is high, RESET goes low for at least eight cycles of ϕ_1 , ϕ_2 during which all internal registers are cleared and interrupt bit (I) in CC is set.
2. Data at FFFE loads into PCH.
3. Data at FFFF loads into PCL.
4. PC contents go out on ADRS bus during ϕ_1 .
5. Contents of cell addressed enters instruction register during ϕ_2 and is decoded as first instruction.
6. If two or more byte instruction, additional bytes enter MPU for execution. If not, go to next step.
7. After execution, step 5 is repeated for subsequent instructions.

IRQ SEQUENCE

1. If bit "I" in condition code register is not set ($I = 0$) and $\overline{\text{IRQ}}$ goes low for at least one ϕ_2 cycle, the $\overline{\text{IRQ}}$ sequence will be entered.
2. After completion of the current instruction, internal registers PC, X, A, B and CC will be stored in RAM at the address indicated by the stack pointer in descending locations (7 bytes in all).
3. The $\overline{\text{IRQ}}$ mask (bit I = 1) is set.
4. Data at FFF8 gets loaded into PCH.
5. Data at FFF9 gets loaded into PCL.
6. PC contents go out on address bus during ϕ_1 .
7. Contents of call addressed enters instruction register during ϕ_2 and is decoded as first instruction of interrupt routine.
8. If it is a more than 1 byte instruction, additional bytes enter MPU for execution. If not, go to next step.
9. After execution, step 5 is repeated for subsequent instructions. This loop is repeated until the instruction "RTI" is executed.

NMI SEQUENCE

1. If $\overline{\text{NMI}}$ goes low for at least one ϕ_2 cycle, the MPU will wait for completion of current instruction.
2. The internal registers PC, X, A, B and CC will then be stored in RAM at the address indicated by the stack pointer in descending locations (7 bytes in all).
3. The $\overline{\text{IRQ}}$ (bit I = 1) mask is set.
4. Data at FFFC is loaded into PCH.
5. Data at FFFD is loaded into PCL.
6. PC contents go out on AD RS bus during ϕ_1 .
7. Contents of cell addressed enters instruction register during ϕ_2 and is decoded as first instruction of NMI subroutine.
8. If two or more byte instruction, additional bytes enter MPU for execution. If not, go to next step.
9. After execution, Step 5 is repeated for subsequent instructions. This loop is repeated until the instruction "RTI" is executed.

RTI EXECUTION

1. The contents of the stack are loaded back into the MPU. (unwinds)
2. The contents of the PC go out on the address bus to fetch the first byte of the next instruction.

SWI INSTRUCTION

1. Contents of the MPU registers PC, 1X, ACCA, ACCB and CC are stored in RAM at the address indicated by the stack pointer in descending location (7 bytes in all).
2. The $\overline{\text{IRQ}}$ mask (bit I = 1) is set.
3. Data at FFFA gets loaded into PCH.
4. Data at FFFB gets loaded into PCL.
5. PC contents go out on address bus during ϕ_1 .
6. Contents of cell addressed enters instruction register during ϕ_2 and is decoded as first instruction of SWI subroutine.
7. If it is a more than one byte instruction, additional bytes enter MPU for execution. If not, go to next step.
8. After execution, Step 6 is repeated for subsequent instructions. This loop is repeated until the instruction "RTI" is executed.

Number Systems

Everyone is quite familiar with the base 10 number system i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8, & 9, since this is the system we all use day to day. Let us review a typical number, say 2743, and see what it really means. The least significant digit (LSD) is 3 and the most significant digit (MSD) is 2. Since we are talking about a base 10 number, the number 2743 really is $3 \times 10^0 + 4 \times 10^1 + 7 \times 10^2 + 2 \times 10^3 = 3 \times 1 + 4 \times 10 + 7 \times 100 + 2 \times 1000$
 $= 3 + 40 + 700 + 2000$
 $= 2743.$

In digital computers, base 10 numbers are represented in binary form, i.e. 1's & 0's. Lets take a base 10 number and convert it to a binary (base 2) number. A method of doing this is known as "repeated division by 2". The base 10 number of 47 is converted to binary as shown below:

$2 \overline{) 23}$	R=1	
$2 \overline{) 11}$	R=1	
$2 \overline{) 5}$	R=1	
$2 \overline{) 2}$	R=1	
$2 \overline{) 1}$	R=0	
$2 \overline{) 0}$	R=1	

↑

101111_2

Converting 101111_2 back to our base 10 number is done in the same manner as above.

$$\begin{aligned}
 101111_2 &= 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 \\
 &= 1 \times 1 + 1 \times 2 + 1 \times 4 + 1 \times 8 + 0 + 1 \times 32 \\
 &= 1 + 2 + 4 + 8 + 0 + 32 \\
 &= 47_{10}
 \end{aligned}$$

In general, converting from a number in any base to a number in base 10 is accomplished as follows:

$$(A_0 B^0 + A_1 B^1 + A_2 B^2 + A_3 B^3 + A_4 B^4 + \dots + A_n B^n)$$

where B is the base of the number system and A is the particular digit in the original number corresponding to its position to the left of the decimal point. On the example just completed, (101111). $A_0 = 1$, $A_1 = 1$, $A_2 = 1$, $A_3 = 1$, $A_4 = 0$, & $A_5 = 1$ and $B = 2$ (base 2).

Another base which is very convenient in digital computers is base 8, since base 8 is really a convenient way of representing base 2. Lets illustrate by converting a base 10 number to base 8 & base 2. Let's convert 61 in base 10 to a number in base 8 and a number in base 2. By continuous division:

$8 \overline{) 61}$	R=5	↑ 75 ₈
$8 \overline{) 7}$	R=7	
<hr/>		
$2 \overline{) 61}$	R=1	↑ 111101 ₂
$2 \overline{) 30}$	R=0	
$2 \overline{) 15}$	R=1	
$2 \overline{) 7}$	R=1	
$2 \overline{) 3}$	R=1	
$2 \overline{) 1}$	R=1	
$2 \overline{) 0}$	R=0	

First lets prove that 75_8 & 111101_2 are really equal to 61_{10} .

$$\begin{aligned} 75_8 &= 5 \times 8^0 + 7 \times 8^1 \\ &= 5 \times 1 + 7 \times 8 \\ &= 5 + 56 \\ &= 61_{10} \end{aligned}$$

$$\begin{aligned} 111101_2 &= 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 \\ &= 1 \times 1 + 0 + 1 \times 4 + 1 \times 8 + 1 \times 16 + 1 \times 32 \\ &= 1 + 0 + 4 + 8 + 16 + 32 \\ &= 61_{10} \end{aligned}$$

Notice that if we take the base 8 number of 75 and convert each digit to base 2, we have the same number as when we converted the base 10 number to base 2. i.e.

Convert 7 to base 2

$$\begin{array}{rcl} 2 \overline{) 7} & R=1 & \\ 2 \overline{) 3} & R=1 & \uparrow \\ 2 \overline{) 1} & R=1 & \end{array} \quad 111_2$$

Convert 5 to base 2

$$\begin{array}{rcl} 2 \overline{) 5} & R=1 & \\ 2 \overline{) 2} & R=0 & \uparrow \\ 2 \overline{) 1} & R=1 & \end{array} \quad 101_2$$

Therefore $75_8 = 111101$ which is the same pattern of 1's & 0's as we got from converting from base 10 to base 2. What this really says that it is easier to convert any base 10 number to base 8 by continuous division, and then convert each digit of the base 8 number to base 2. Let's look at another example. Convert 183_{10} to base 8 & to base 2.

$8 \overline{) 183}$	R=7	
$8 \overline{) 22}$	R=6	\uparrow 267_8
$8 \overline{) 0}$	R=2	
<div style="display: flex; justify-content: space-between; width: 100%;"> $2 \overline{) 183}$ R=1 </div>		
$2 \overline{) 91}$	R=1	
$2 \overline{) 45}$	R=1	\uparrow 10110111_2
$2 \overline{) 22}$	R=0	
$2 \overline{) 11}$	R=1	
$2 \overline{) 5}$	R=1	
$2 \overline{) 2}$	R=0	
$2 \overline{) 1}$	R=1	

$$\begin{aligned}
 267_8 &= 7 \times 8^2 + 6 \times 8^1 + 2 \times 8^0 \\
 &= 7 \times 64 + 6 \times 8 + 2 \times 1 \\
 &= 448 + 48 + 2 \\
 &= 498
 \end{aligned}$$

to convert

267_8 directly to base 2, we convert each base 8 digit separately.

$$\begin{array}{rcl}
 2 & 2 \overline{) 2} & R=0 \\
 & 2 \overline{) 0} & \uparrow R=1 \quad 10 \\
 & 2 \overline{) 1} & \\
 \hline
 6 & 2 \overline{) 3} & R=0 \\
 & 2 \overline{) 1} & \uparrow R=1 \quad 110 \\
 & 2 \overline{) 3} & \\
 & 2 \overline{) 0} & R=1 \\
 & 2 \overline{) 1} & \\
 \hline
 7 & 2 \overline{) 3} & R=1 \\
 & 2 \overline{) 1} & \uparrow R=1 \quad 111 \\
 & 2 \overline{) 3} & \\
 & 2 \overline{) 0} & R=1 \\
 & 2 \overline{) 1} &
 \end{array}$$

therefore $2_8 = 10_2$, $6_8 = 110_2$, & $7_8 = 111_2$ and
 $267_8 = 10110111_2$

Digital computers are designed to use binary numbers in their working registers. The working registers vary in number of bits depending on the manufacturer. The Motorola M6800 micro-processor utilizes, in general, 8 bit words (or registers). This leads to another number base, not yet mentioned, of hexadecimal. Hexadecimal is really a base 16 number system and can be handled in exactly the same manner as base 8 or base 2. In hexadecimal, four bits (in binary) represents one hexadecimal number. Thus , an eight bit register can be represented by a hex number of 2 digits long. To illustrate, lets assume we have the number of 147_8 in an eight bit register. This in binary form is 01100111 . If this bit pattern is divided into 2- four bit words of 0110 & 0111, then in hex, 147_8 can be represented as 67_{16} . To prove both are equal, lets convert both back to their base 10 number.

$$\begin{aligned}
 147_8 &= 7 \times 8^0 + 4 \times 8^1 + 1 \times 8^2 \\
 &= 7 \times 1 + 4 \times 8 + 1 \times 64 \\
 &= 7 + 32 + 64 \\
 &= 103_{10}
 \end{aligned}$$

$$\begin{aligned}
 67_{16} &= 7 \times 16^0 + 6 \times 16^1 \\
 &= 7 \times 1 + 6 \times 16 \\
 &= 7 + 96 \\
 &= 103_{10}
 \end{aligned}$$

As you probably have wondered by now, how do we represent these hex (base 16) numbers above 9? Here is the base 16 number compared with its equivalent base 10 number.

<u>Base 10</u>	<u>Base 16</u>
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C

13

D

14

E

15

F

To convert any base 10 number to hex (base 16) you may convert it to base 8 first, then represent the base 8 number with its binary representation. By taking the binary representation of the number and grouping the bits from right to left in groups of four which are then represented in hex per the above table. Or one may convert any base 10 number to hex by our continuous division rule as before. Lets convert 825_{10} to hex.

$$\begin{array}{rcl}
 16 \overline{) 825} & R=9 & \\
 16 \overline{) 51} & R=3 & \uparrow \\
 16 \overline{) 3} & R=3 & \\
 \end{array}
 \quad 339_{16}$$

therefore $825_{10} = 339_{16}$

to convert 339_{16} back to our base 10 number,

$$\begin{aligned}
 339_{16} &= 9 \times 16^0 + 3 \times 16^1 + 3 \times 16^2 \\
 &= 9 \times 1 + 3 \times 16 + 3 \times 256 \\
 &= 9 + 48 + 768 \\
 &= 825_{10}
 \end{aligned}$$

To show the relationship between hex, binary, and octal, lets convert 825_{10} to octal & to binary and then back to hex.

825₁₀ to octal

$$8 \overline{) 825} \begin{array}{r} 103 \\ 8 \end{array}$$

R=1

$$8 \overline{) 103} \begin{array}{r} 12 \\ 8 \end{array}$$

R=7

1471₈

$$8 \overline{) 12} \begin{array}{r} 1 \\ 8 \end{array}$$

R=4

$$8 \overline{) 0} \begin{array}{r} 0 \\ 1 \end{array}$$

R=1

825₁₀ to binary

$$2 \overline{) 825} \begin{array}{r} 412 \\ 2 \end{array}$$

R=1

$$2 \overline{) 412} \begin{array}{r} 206 \\ 2 \end{array}$$

R=0

$$2 \overline{) 206} \begin{array}{r} 103 \\ 2 \end{array}$$

R=0

$$2 \overline{) 103} \begin{array}{r} 51 \\ 2 \end{array}$$

R=1

$$2 \overline{) 51} \begin{array}{r} 25 \\ 2 \end{array}$$

R=1

1100111001₂

$$2 \overline{) 25} \begin{array}{r} 12 \\ 2 \end{array}$$

R=1

$$2 \overline{) 12} \begin{array}{r} 6 \\ 2 \end{array}$$

R=0

$$2 \overline{) 6} \begin{array}{r} 3 \\ 2 \end{array}$$

R=0

$$2 \overline{) 3} \begin{array}{r} 1 \\ 2 \end{array}$$

R=1

$$2 \overline{) 1} \begin{array}{r} 0 \\ 1 \end{array}$$

R=1

$$825_{10} = 1471_8$$

$$= 1 \times 8^0 + 7 \times 8^1 + 4 \times 8^2 + 1 \times 8^3$$

$$= 1 \times 1 + 7 \times 8 + 4 \times 64 + 1 \times 512$$

$$= 1 + 56 + 256 + 512$$

$$= 825_{10}$$

$$\begin{aligned}
825_{10} &= 1100111001_2 \\
&= 1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + 0 \times 2^7 + 1 \times 2^8 + 1 \times 2^9 \\
&= 1 \times 1 + 0 + 0 + 1 \times 8 + 1 \times 16 + 1 \times 32 + 0 + 0 + 1 \times 256 + 1 \times 512 \\
&= 1 + 0 + 0 + 8 + 16 + 32 + 0 + 0 + 256 + 512 \\
&= 825_{10}
\end{aligned}$$

Or taking 1471_8 and representing each digit by its binary representation, we get $1=001$, $4=100$, $7=111$ & $1=001$ which when put together equal 001100111001 . Notice this is the same bit pattern as when we converted from base 10 to base 2. Now if we group this into three groups of four bits and then convert each group to its hex counterpart, we will have the number of 825_{10} represented in hex. $001100111001 = 0011\ 0011\ 1001 = 339_{16}$. Notice this agrees with the result when we converted directly to hex from one base 10 number.

In summary, let's take the situation when an MPU 6800 8 bit register contains all 1's.

$$\begin{aligned}
11111111 &= 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^7 \\
&= 1 \times 1 + 1 \times 2 + 1 \times 4 + 1 \times 8 + 1 \times 16 + 1 \times 32 + 1 \times 64 + 1 \times 128 \\
&= 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 \\
&= 255_{10}
\end{aligned}$$

or

$$\begin{aligned}
11111111 &= 11\ 111\ 111 \\
&= 3\ 7\ 7_8 = 3 \times 8^0 + 7 \times 8^1 + 3 \times 8^2 \\
&= 3 \times 1 + 7 \times 8 + 3 \times 64 \\
&= 3 + 56 + 192 \\
&= 255_{10}
\end{aligned}$$

or

$$11111111 = 1111 \ 1111$$

$$F_{16} = 15 \times 16^0 + 15 \times 16^1$$

$$= 15 \times 1 + 240$$

$$= 15 + 240$$

$$= 255_{10}$$

Conversion Chart

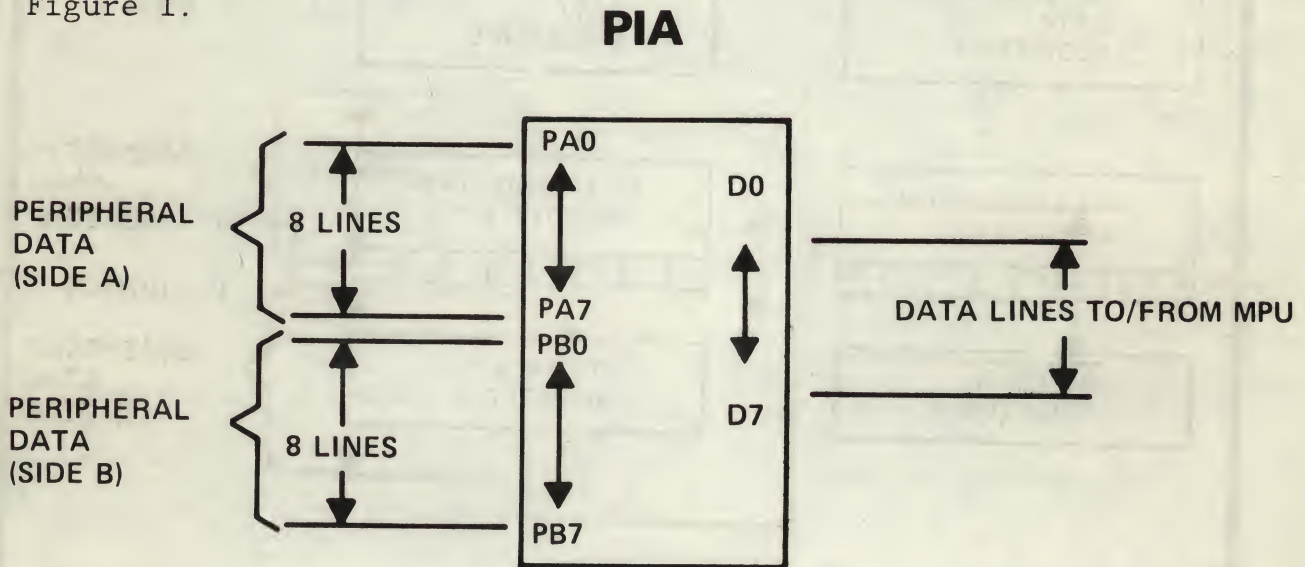
<u>Decimal</u>	<u>Octal</u>	<u>hexadecimal</u>	<u>binary</u>
0	0	0	0000 0000
1	1	1	0000 0001
2	2	2	0000 0010
3	3	3	0000 0011
4	4	4	0000 0100
5	5	5	0000 0101
6	6	6	0000 0110
7	7	7	0000 0111
8	10	8	0000 1000
9	11	9	0000 1001
10	12	A	0000 1010
11	13	B	0000 1011
12	14	C	0000 1100
13	15	D	0000 1101
14	16	E	0000 1110
15	17	F	0000 1111
16	20	10	0001 0000
17	21	11	0001 0001
18	22	12	0001 0010
19	23	13	0001 0011
20	24	14	0001 0100
21	25	15	0001 0101

22	26	16	0001 0110
23	27	17	0001 0111
24	30	18	0001 1000
25	31	19	0001 1001
26	32	1A	0001 1010
27	33	1B	0001 1011
28	34	1C	0001 1100
29	35	1D	0001 1101
30	36	1E	0001 1110
31	37	1F	0001 1111
32	40	20	0010 0000
33	41	21	0010 0001
34	42	22	0010 0010
35	43	23	0010 0011
36	44	24	0010 0100
37	45	25	0010 0101
38	46	26	0010 0110
39	47	27	0010 0111
40	50	28	0010 1000

Peripheral Interface Adapter (PIA) - MC6820

The Peripheral Interface Adapter (PIA) is a means used to interface peripheral equipment with the microprocessing unit (MPU). The PIA communicates with the MPU via an eight bit bi-directional data bus, three chip selects, two register selects, two interrupt request lines, one read/write line, an enable line, and a reset line. These will be discussed in detail later.

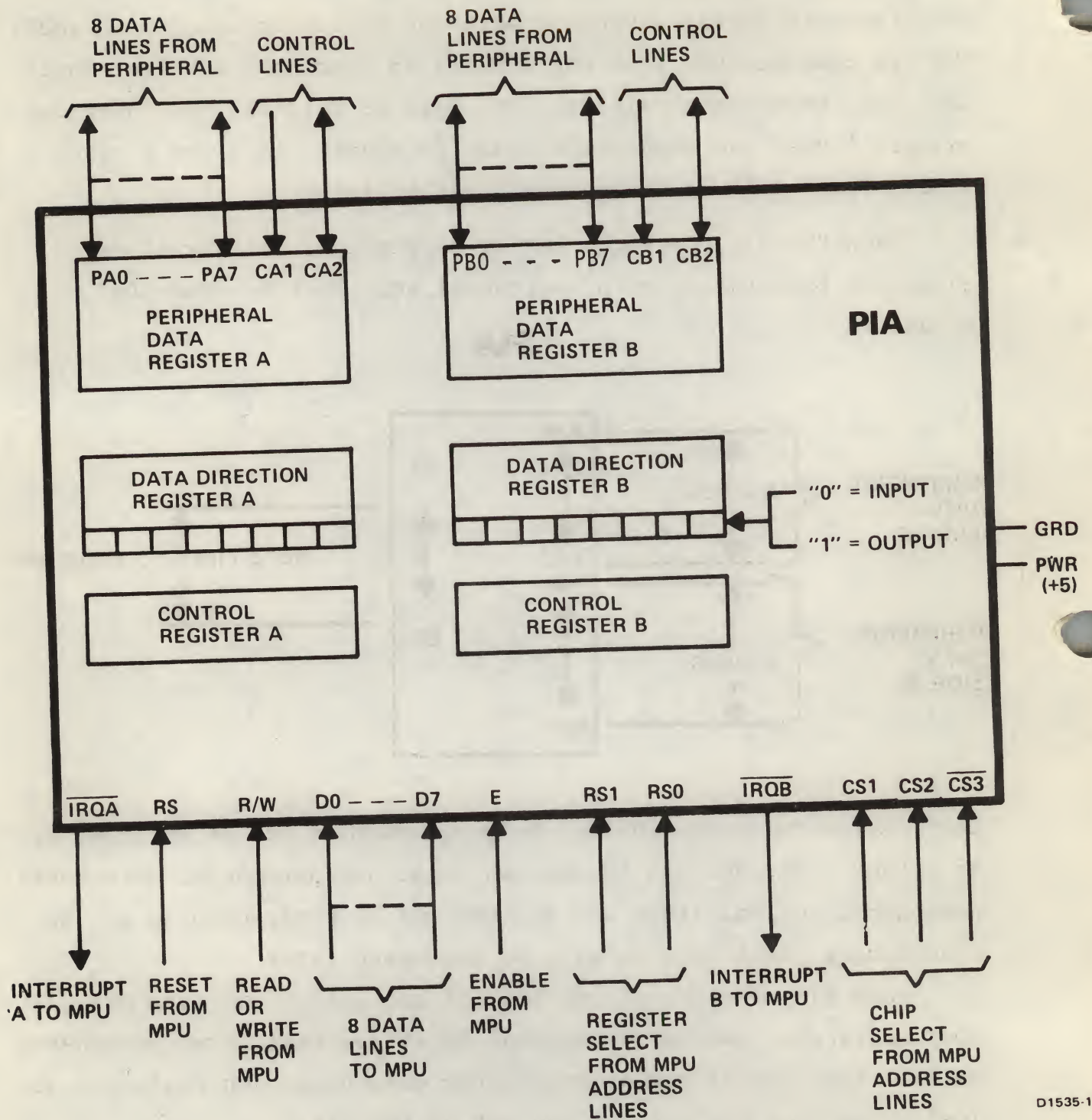
Each PIA has two eight bit bi-directional peripheral data buses for interfacing with peripheral equipment as shown in Figure 1.



D1534

Each Peripheral data line may be programmed to act as an input or an output. In addition to the two eight bit peripheral data buses, peripheral control lines CA2 and CB2 may be programmed to act as a peripheral data line as will be discussed later.

Each PIA consists of two control registers, two data direction registers, and two peripheral interface registers (peripheral data). The control registers and the data direction registers are used to control the data in and out of the PIA.



A. Peripheral Data Lines PA0 thru PA7

Each of these 8 data lines which interface with the outside world can be programmed to act as either an input or an output. This is accomplished by setting a "1" in the corresponding bit in the Data Direction Register (DDR) if the line is to be an output or a "0" in the DDR if the line is to be an input. When the data in the peripheral data lines are read into the MPU by a load instruction, those lines which have been designated as input lines (0 in DDR) will be gated directly to the data bus and into the register selected in the MPU. In the input mode, each line represents a maximum of one standard TTL load.

On the other hand, when an output data instruction (STA A PIA) is executed, data will be transferred via the data bus to the peripheral data register. A "1" output will cause a "high" on the corresponding data line and a "0" output will cause a "low" on the corresponding data line. Data in Peripheral Register A that have been programmed as outputs may be read by an MPU "LDA A from PIA" instruction. If the voltage is above 2 volts for a logic "1" or below .8 volts for a logic "0", the data will agree with that data outputted. However, if these output lines have been loaded such that they do not meet the levels for logic "1", the data read back into the MPU may differ from the data stored in the PIA Peripheral Register A.

B. Peripheral Data Lines PB0 thru PB7

The 8 data lines which interface with the outside world on the B side may also be programmed to act either as an input or as an output. This is also accomplished by setting a "1" in the corresponding bit in the Data Direction Register (DDR) if the line is to be an output or a "0" in the DDR if the line is to be

an input. The output buffers driving these lines have three state capability, allowing them to enter a high impedance state when the peripheral data line is used as an input. Data in Peripheral Register B that have been programmed as outputs may be read by an MPU "LDA A from PIA" instruction even though the lines have been programmed as outputs. If the line has been programmed as an output ("1"), reading the line will indicate a logic "1" due to buffering between the register and the output pin.

C. Data Lines (D0-D7)

The 8 bi-directional data lines permit transfer of data to/from the PIA and the MPU. The MPU receives data from the outside world from the PIA via these 8 data lines or sends data to the outside world through the PIA's via the 8 data lines. The data bus output drivers are three state devices that remain in the high impedance (off) state except when the MPU performs a PIA read operation.

D. Chip Select Lines (CS1, CS2, $\overline{\text{CS3}}$)

These are the lines which are tied to the address lines of the MPU. It is through these lines that a particular PIA is selected (addressed). For selection of a PIA, the CS1 and CS2 lines must be high and the $\overline{\text{CS3}}$ must be low. After the chip selects have been addressed, they must be held in that state for the duration of the E (enable) pulse, which is the only timing signal supplied by the MPU to the PIA. This enable pulse (E) is normally the $\phi 2$ clock. One of the address lines should be ANDed with the VMA line with this output tied to a chip select.

E. Enable Line (E)

The enable pulse (E) is the only timing signal that is supplied to the PIA by the MPU. Timing on all other signals is referenced to the leading or trailing edges of the E pulse.

F. Reset Line (RS)

This line is used to reset all registers in the PIA to a logical zero. This would be used primarily during a reset or power on operation. This line is normally in the high state. The transition of high to low to high resets all registers in the PIA.

G. Read/Write Line (R/W)

This signal is generated by the MPU to control the direction of the data transfers on the Data Bus. A low state on the PIA Read/Write line enables the input buffers and data is transferred from the MPU to the PIA (MPU write) on the E signal if the device has been selected. A high on the Read/Write line sets up the PIA for a transfer of data to the data bus (MPU read). The PIA output buffers are enabled when the proper address & the enable pulse are present thus transferring data to the MPU.

H. Interrupt Request Lines ($\overline{\text{IRQA}}$ & $\overline{\text{IROB}}$)

These lines are used to interrupt the MPU either directly or indirectly through interrupt priority circuitry. These lines are "open source" (no load device on the chip) and are capable of sinking a current of 1.6 ma from an external source. This permits all interrupt request lines to be tied together in a "wired OR" configuration. Interrupts are serviced by a software routine that sequentially reads & tests, on a prioritized basis, the two control registers in each PIA for interrupt flag bits (Bit 6 & 7) that are set. Discussion on the control registers & how the flag bits get set will follow. When the MPU reads the Peripheral Data Register, the Interrupt Flag (Bit 6 or Bit 7) is cleared & the Interrupt Request is cleared.

These request lines ($\overline{\text{IRQA}}$ & $\overline{\text{IRQB}}$) are active low.

I. Interrupt Input Lines (CA1 & CB1)

These lines are input only to the PIA and set the interrupt flag (Bit 7) of the control registers in the PIA. Discussion of these lines in conjunction with the control register will follow.

J. Peripheral Control Line (CA2)

This line can be programmed to act either as an interrupt input or as a peripheral output. As an output, this line is compatible with standard TTL and as an input represents one standard TTL load. The function of this line is programmed with Control Register A (Bits 3,4,&5).

K. Peripheral Control Line (CB2)

This line may also be programmed to act as an interrupt input or as a peripheral output. As an input, this line has greater than 1 megohm input impedance & is compatible with standard TTL. As an output, it is compatible with standard TTL and may also be used as a source of up to 1 millamp at 1.5 volts to directly drive the base of a transistor switch. The function of this line is programmed with Control Register B (Bits 3,4, & 5).



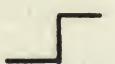

CONTROL REGISTER A (CRA)

7	6	5	4	3	2	1	0
IRQA1	IRQA?	CA2 CONTROL			DDRA	CA1 CONTROL	

CA1 Control (Bit 0 & 1)

Peripheral control line CA1 is an input only line which may be used to cause an interrupt by setting the interrupt flag IRQA1 (Bit 7) of control register A(CRA). Bits 0 and 1 of CRA are used to determine how the interrupt is to be handled.

After the MPU reads Peripheral Data Register A, the IRQA1 (Bit 7) will be cleared.

Transition of interrupt input line CA1	Status of Bit 1 in CRA	Status of Bit 0 in CRA	IRQA1 (Interrupt flag) Bit 7 of CRA	STATUS OF IRQA LINE (MPU INTERRUPT REQUEST)
	0	0	1	MASKED (Remains High)
	0	1	1	GOES LOW (Processor Interrupted)
	1	0	1	MASKED (Remains High)
	1	1	1	GOES LOW (Processor Interrupted)

(All other combinations of CA1 transition and status of bit 0 and bit 1 will be ignored)

Data Direction Access Control (DDRA)-(Bit 2)

This bit, in conjunction with the register select lines RS0 & RS1), is used to select either the Peripheral Data Register or the Data Direction Register. To address the A side control register, RS1 is set to a logic "0" and RS0 is set to a logic "1".

RS1	RS0	CRA (BIT2)	Register Selected
0	0	1	Peripheral Data Reg. A
0	0	0	Data Dir. Reg. A
0	1	-	Control Reg. A


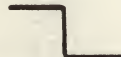


CONTROL REGISTER B (CRB)

7	6	5	4	3	2	1	0
IRQB1	IRQB2	CB2 CONTROL			DDRB	CB1 CONTROL	

CB1 Control (Bit 0 & 1)

Peripheral control line CB1 is an input only line which may be used to cause an interrupt by setting the interrupt flag IRQB1 (Bit 7) of control register B (CRB). Bits 0 and 1 of CRB are used to determine how the interrupt is to be handled.

After the MPU reads Peripheral Data Register B, the IRQB1 (Bit 7) will be cleared.

<u>Transition of interrupt input line CB1</u>	<u>Status of Bit 1 in CRB</u>	<u>Status of Bit 0 in CRB</u>	<u>IRQB1 (Interrupt flag) Bit 7 of CRB</u>	<u>Status of IRQB Line (MPU Interrupt Request)</u>
	0	0	1	MASKED (Remains High)
	0	1	1	GOES LOW (Processor Interrupted)
	1	0	1	MASKED (Remains High)
	1	1	1	GOES LOW (Processor Interrupted)
(All other combinations of CB1 transition and status of bit 0 and bit 1 will be ignored)				





Data Direction Access Control (DDRB)-Bit 2)

This bit, in conjunction with the register select lines (RS0 & RS1), is used to select either the Peripheral Data Register or the Data Direction Register. To address the B side control register, RS1 is set to a logic "1" and RS0 is set to a logic "1".

<u>RS1</u>	<u>RS0</u>	<u>CRB (BIT2)</u>	<u>Register Selected</u>
1	0	1	Peripheral Data Reg. B
1	0	0	Data Dir. Reg. B
1	1	—	Control Reg. B

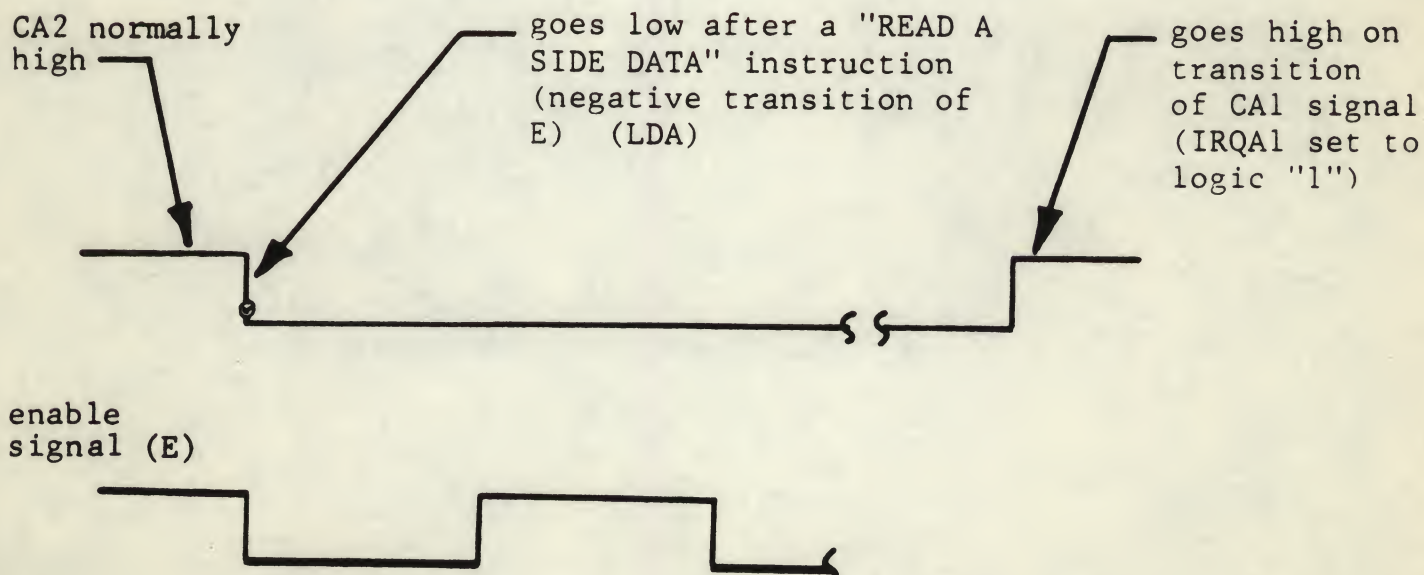
CA2 Control (Bit 3,4, & 5 of CRA)

This line, in addition to generating an interrupt signal, may also be used as an additional output signal. Bits 3,4, & 5 of the control register determine the function of this line.

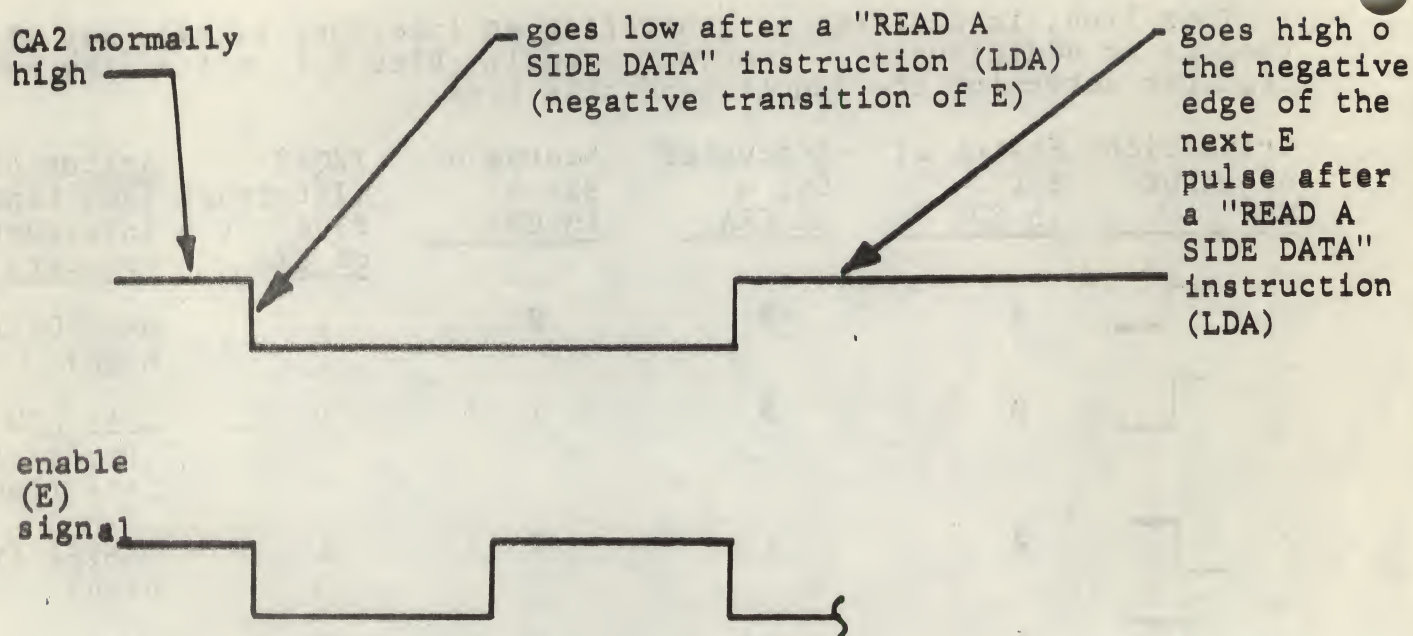
<u>Transition of input CA2</u>	<u>Status of Bit 5 in CRA</u>	<u>Status of Bit 4 in CRA</u>	<u>Status of Bit 3 in CRA</u>	<u>IRQA2 (Interrupt flag) bit 6 of CRA</u>	<u>Status of IRQA Line (MPU interrupt request)</u>
	0	0	0	1	MASKED (remains high)
	0	0	1	1	GOES LOW (Processor interrupted)
	0	1	0	1	MASKED (remains high)
	0	1	1	1	GOES LOW (processor interrupted)
(All other combinations of CA2 transition and status of bit 3 and bit 4 will be ignored)					

CA2 Used As An Output

If bit 5 of CRA is set to a logic "1", CA2 is designated as an output. The four options utilizing CA2 as an output are shown below. In all four options the IRQA2 flag (bit 6 of CRA) remains clear and the IRQA interrupt request line remains high.

Bit 5, 4, 3 of CRA = 100 (Handshake Mode)

BIT 5,4, 3 of CRA = 101



BIT 5,4,3 of CRA = 110

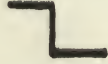

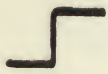

CA2 will always be low with Bits 5,4, & 3 equal to 110

Bit 5,4,3 of CRA = 111

CA2 will always be high with Bits 5, 4, and 3 equal to 111.

CB2 Control (Bit 3,4, & 5 of CRB)

This line, in addition to generating an interrupt signal, may also be used as an additional output signal. Bits 3,4, & 5 of the control register determine the function of this line.

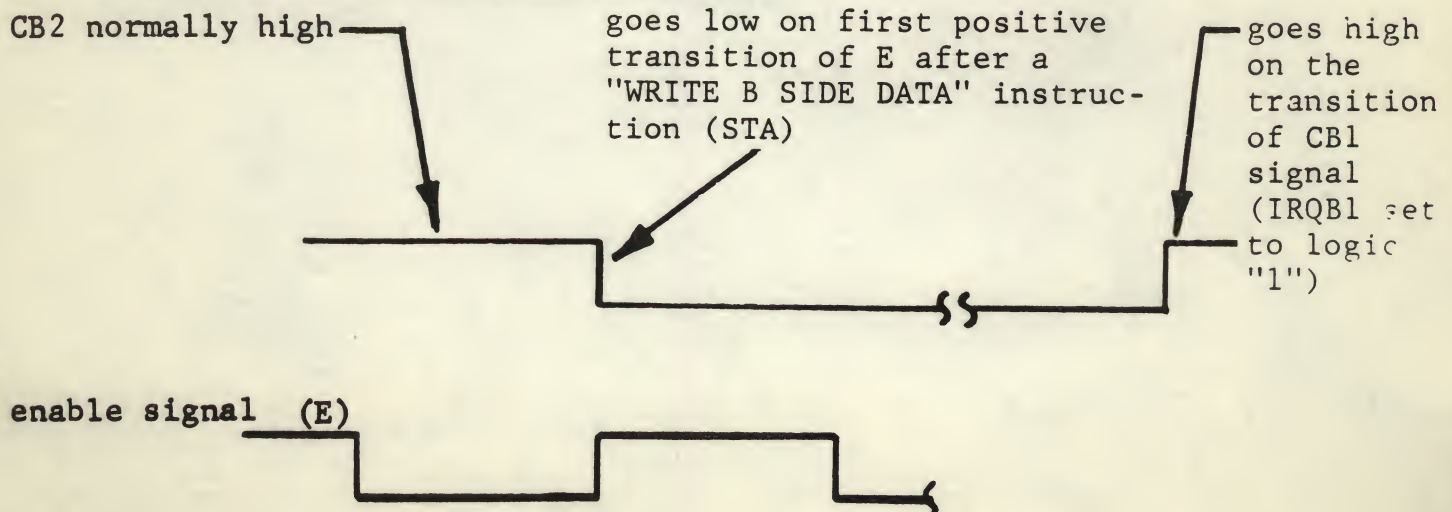
<u>Transition of input CB2</u>	<u>Status of Bit 5 in CRB</u>	<u>Status of Bit 4 in CRB</u>	<u>Status of Bit 3 in CRB</u>	<u>IROB2 (Interrupt flag) bit 6 of CRB</u>	<u>Status of IRQB Line (MPU interrupt request)</u>
	0	0	0	1	MASKED (remains high)
	0	0	1	1	GOES LOW processor interrupted
	0	1	0	1	MASKED (remains high)
	0	1	1	1	GOES LOW (processor interrupted)

(All other combinations of CB2 transition and status of bit 3 and bit 4 will be ignored.)

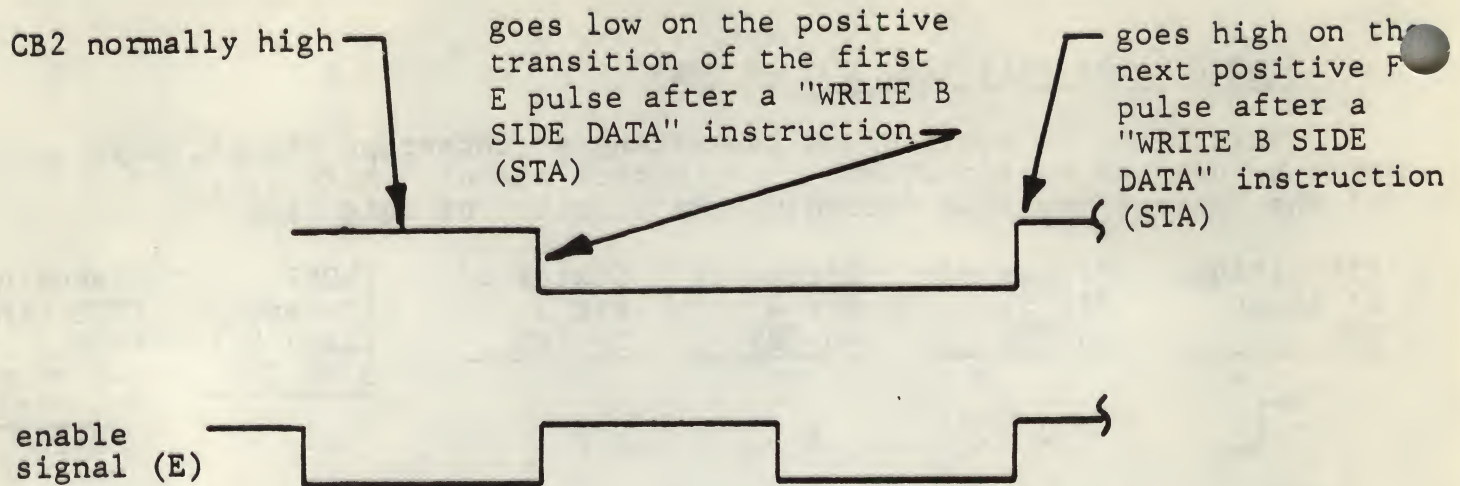
CB2 Used as an Output

If bit 5 of CRB is set to a logic "1", CB2 is designated an output. The four options utilizing CB2 as an output are shown below. In all four options, the IRQB2 flag (bit 6 of CRB) remains clear and the IRQB interrupt request line remains high

Bit 5, 4, 3 of CRB = 100 (Handshake Mode)



Bit 5,4,3 of CRB = 101



Bit 5,4,3 of CRB = 110

CB2 will always be low with Bits 5,4, & 3 of CRB is equal to 110.

Bit 5,4,3 of CRB = 111

CB2 will always be high when bits 5,4, & 3 of CRB is equal to 111.

SUMMARY OF PIA CONTROL REGISTERS:a) Register selects RS0 & RS1

If RS1 is set to a logic "0", then "A" side is selected

If RS1 is set to a logic "1", then the "B" side is selected.

If RS0 is set to a logic "0", and CRA (or CRB) Bit 2 is set to a logic "1", the peripheral data register is selected.

If RS0 is set to a logic "0", and CRA (or CRB) Bit 2 is set to a logic "0", then the data direction register is selected.

If RS0 is set to a logic "1", the control register is selected.

b) CA1 or CB1 Interrupt Line

If bit 0 of CRA (or CRB) is set to a logic "0", all interrupts caused by CA1 (or CB1) are disallowed by the PIA.

c) CA2 or CB2 Interrupt Line

If bit 3 of CRA (or CRB) is set to a logic "0", all interrupts caused by CA2 (or CB2) are disallowed by the PIA. If bit 5 of CRA (or CRB) is set to a logic "1", then the CA2 (or CB2) line is used as an output line per previous table.

Summary of Control Registers CRA & CRB

Control Registers CRA & CRB have total control of CA1, CA2, CB1, and CB2 lines. The status of eight bits of the control registers may be read into the MPU. However, the MPU can only write into bit 0 thru bit 5 (6 bits), since bit 6 & bit 7 are set only by CA1, CA2, CB1, or CB2.

Addressing PIA's

Before addressing PIA's, the Data Direction (DDR) must first be loaded with the bit pattern that defines how each line is to function i.e. as an input or an output. A logic "1" in the Data Direction Register defines the corresponding line as an output while a logic "0" defines the corresponding line as an input. Since the DDR and the Peripheral Data Lines have the same address, the control register bit 2 determines which register is being addressed. If bit 2 in the control register is a logic "0", then the DDR is addressed. If bit 2 in the control register is a logic "1", the Peripheral Data Register is addressed. Therefore, it is essential that the DDR be loaded first before setting bit 2 of the control register.

Example: Given a PIA with an address of 4004, 4005, 4006, & 4007. 4004 is the address of the A side Peripheral Interface Register. 4005 is the address of the A side control register. 4006 is the address of the B side Peripheral Interface Register. 4007 is the address of the B side control register. On the A side, bit 0, 1, 2, & 3 will be defined as inputs while bit 4,5,6 & 7 will be used as outputs. On the B side, all lines will be used as outputs.

The program to accomplish the above is as follows

```
PIA1AD = 4004
PIA1AC = 4005
PIA1BD = 4006
PIA1BC = 4007
```

```
1. LDA A #% 11110000      (4 inputs, 4 outputs)
2. STA A PIA1AD           (loads A DDR)
3. LDA A #% 11111111      (All outputs)
4. STA A PIA1BD           (Loads B DDR)
5. LDA A #% 00000100      (sets bit 2)
6. STA A PIA1AC           (Bit 2 set in A contr. reg)
7. STA A PIA1BC           (Bit 2 set in B contr. reg)
```

Statement 2 addresses the DDR since the Control Register (Bit 2) has not been loaded. Statement 6 & 7 loads the control registers with bit 2 set, so addressing PIA1AD or PIA1BD accesses the Data Register.

ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER (ACIA) - MC6850

The Asynchronous Communications Interface Adapter (ACIA) is a means used to receive and transmit up to eight bits of data for serial data communications. The ACIA communicates with the MPU via an eight bit bi-directional data bus, three chip select lines, one register select line, one interrupt request line, an enable line, and one read/write line.

The ACIA has four registers which may be addressed by the MPU. The Status Register (SR) and the Receiver Data Register (RDR) are "read only" registers in that the MPU cannot write into two registers. The transmit Data Register (TDR) and the Control Register (CR) are "write only" registers in that the MPU cannot read from these registers.

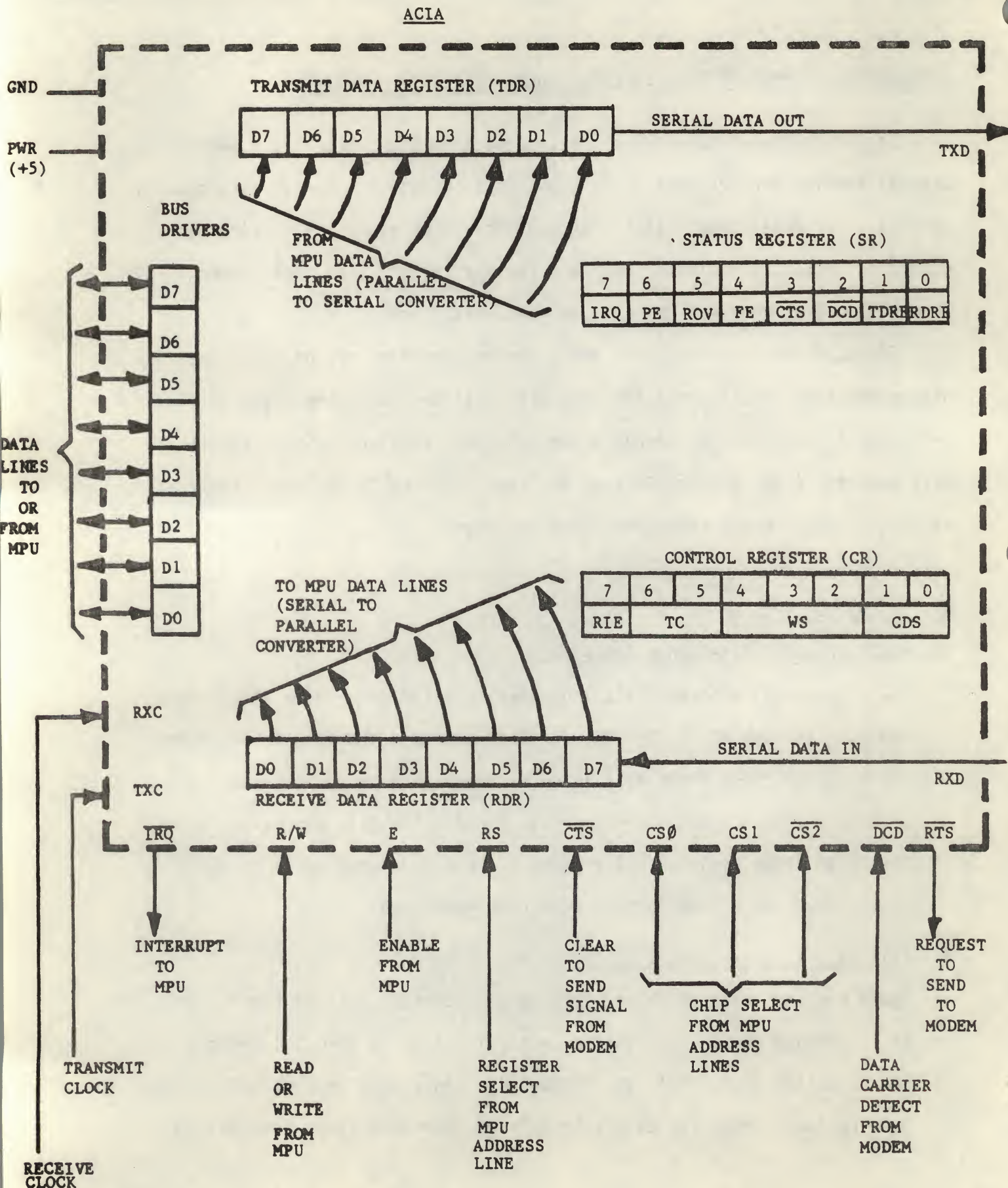
MPU INTERFACE LINES

A. Bi-Directional Data Lines (D0 - D7)

The eight bi-directional data lines permit transfer of data to and from the ACIA and the MPU. The MPU receives data from the outside world from the ACIA via these eight data lines or sends data to the outside world through the ACIA's via the eight data lines. The data bus output drivers are three state devices that remain in the high impedance (off) state except when the MPU performs a ACIA read operation.

B. Chip Select Lines (CS0, CS1, CS2)

These are the lines which are tied to the address lines of the MPU. It is through these lines that a particular ACIA is selected (addressed). For selection of an ACIA, the CS0 and CS1 lines must be high and the CS2 must be low. After the chip selects have been addressed, they must be



held in that state for the duration of the E enable pulse, which is the only timing signal supplied by the MPU to the ACIA .

C. Enable Signal (E)

The enable pulse is a high impedance TTL compatible input from the MPU that enables the ACIA input or output buffers and clocks data to or from the ACIA.

D. Read/Write Line (R/W)

The Read/Write line is a high impedance TTL compatible input that is used to control the direction of data flow between the ACIA's eight bit parallel data bus and the MPU. When Read/Write is high (MPU read), the ACIA output driver is turned on and a selected register is read by the MPU. When the Read/Write line is low (MPU write), the ACIA output driver is turned off and the MPU writes into a selected register. Thus, the Read/Write signal, in conjunction with the register select line, is used to select the registers within the ACIA that are read only.

<u>Register Select RS</u>	<u>Read/Write (R/W)</u>	<u>ACIA Register Selected</u>	<u>MPU Read or Write</u>
0	0	Control	Write
0	1	Status	Read
1	0	Transmit Data	Write
1	1	Receive Data	Read

E. Register Select (RS)

The Register Select line is a high impedance TTL compatible input from the MPU that is used to select, in conjunction with the Read/Write line, either the Transmit/Receiver Data Register or the Control/Status register in the ACIA as shown in part D of this section.

F. Interrupt Request Line ($\overline{\text{IRQ}}$)

The Interrupt Request Line is a TTL compatible output line to the MPU that is used to interrupt the MPU upon the occurrence of certain events. This line is active in the low state and remains low as long as the course of the interrupt is present and the appropriate interrupt enable within the ACIA is set.

ACIA REGISTERS

A. Status Register (Read Only)

The Status Register can only be read by the MPU. This register is selected when the Register Select (RS) line is low and the Read/Write (R/W) line is high ($\text{RS} \cdot \text{R/W} = 01$)

STATUS REGISTERS (SR)

7	6	5	4	3	2	1	0
IRQ	PE	ROV	FE	$\overline{\text{CTS}}$	$\overline{\text{DCD}}$	TDRE	RDRF

Bit 0 - Receiver Data Register Full (RDRF)

- "1" - The Receiver Data Register is full. When this bit gets set to a logic "1" indicating the Receiver Data Register is full, the IRQ bit (bit 7) gets set also and remains set until the data is read into the MPU.
- "0" - The Receiver Data Register has been read by the MPU. The non-destructive read cycle clears the RDRF bit although the data in the Receiver Data Register is retained. If the $\overline{\text{DCD}}$ line goes high indicating loss of carrier, the RDRF bit is clamped at logic "0" indicating the contents of the Receiver Data Register are not current.

Bit 1 - Transmit Data Register Empty (TDRE)

"1" - The Transmit Data Register is empty and new data may be transferred.

- IRQ (bit 7 gets set)

"0" - The Transmit Data Register is full

Bit 2 - Data Carrier Detect ($\overline{\text{DCD}}$)

"1" - There is no carrier from the modem. When this bit goes to a logic

"1" - the IRQ bit (bit 7) of the status register gets set and remains set until the MPU reads the Status Register and the Receiver Data Register.

"0" - The carrier from the modem is present.

Bit 3 - Clear to Send ($\overline{\text{CTS}}$)

"1" - The Clear to Send line from the modem is high, thus inhibiting the Transmit Data Register Empty (TDRE) bit. Modem is not ready for data.

"0" - The Clear to Send line from the modem is low. Modem is ready for data.

Bit 4 - Framing Error (FE)

"1" - Framing error indicates that the received character is improperly framed by the start and stop bit and is detected by the absence of the first stop bit. This error indicates a synchronization error, faulty transmission, or a break condition. This error flag is set or reset during the receiver data transfer time. Therefore, this error indicator is present throughout the time that the associated character is available.

"0" - The received character is properly framed.

Bit 5 - Receiver Overrun (ROV)

- "1" - Overrun is an error flag that indicates that one or more characters in the data stream were lost. That is, a character or a number of characters were received but not read from the Receiver Data Register (RDR) prior to subsequent being received. The overrun condition begins at the midpoint of the last bit of the second character received in succession without a read of the RDR having occurred. The Overrun does not occur in the Status Register until the valid character prior to Overrun has been read. Character synchronization is maintained during the Overrun condition. The Overrun indication is reset after the reading of data from the Receive Data Register. Overrun is also reset by the Master Reset.
- "0" - No Receiver Data Overrun have occurred.

Bit 6 - Parity Error (PE)

- "1" - The parity error flag indicates that the number of highs (ones) in the character does not agree with the preselected odd or even parity. Odd parity is defined to be when the total number of ones is odd. The parity error indication will be present as long as the data character is in the RDR. If no parity is selected, then both the transmitter parity generator output and the receiver parity check results are inhibited.
- "0" - No parity error occurred.

Bit 7 - Interrupt Request (IRQ)

- "1" - There is an interrupt in the ACIA. This bit being high causes the $\overline{\text{IRQ}}$ output line to be low. This will be cleared by reading the Status Register and writing into the Transmit Data Register or reading the Receiving Data Register.
- "0" - No interrupt present.

B. Control Register (Write Only)

The Control Register can only be written into by the MPU. This register is selected when the Register Select (RS) line and the Read/Write line are both low ($RS \cdot R/W = 00$)

CONTROL REGISTER (CR)

7	6	5	4	3	2	1	0
R I E	Transmitter Control		Word Select			Counter Divide	

Receiver
Interrupt
Enable

Bit 0 and 1 - Counter Divide Select Bits (CDS)

<u>CR1</u>	<u>CR0</u>	<u>FUNCTION</u>
0	0	$\div 1$
0	1	$\div 16$
1	0	$\div 64$
1	1	Master Reset

Bit 2, 3, 4 - Word Select Bits (WS)

<u>CR4</u>	<u>CR3</u>	<u>CR2</u>	<u>FUNCTION</u>
0	0	0	7 Bit + EP + 2SB
0	0	1	7 Bit + OP + 2SB
0	1	0	7 Bit + EP + 1SB
0	1	1	7 Bit + OP + 1SB
1	0	0	8 Bit + 2SB
1	0	1	8 Bit + 1SB
1	1	0	8 Bit + EP + 1SB
1	1	1	8 Bit + OP + 1SB

EP - Even
OP - Odd
SB - Stop bits

Bit 6 and 5 - Transmitter Control (TC)

<u>CR6</u>	<u>CR5</u>	<u>FUNCTION</u>
0	0	RTS = Low Transmitting Interrupt Disabled (TIE)
0	1	RTS = Low Transmitting Enabled (TIE)
1	0	RTS = High Transmitting Interrupt Disabled (TIE)
1	1	RTS = Low Transmitting Interrupt Disabled (TIE) and transmits a Break level on the transmit data output.

Bit 7 - Receiver Interrupt Enable

"1" - Enables interrupts caused by

- a) Receiver Data Register Full going high
- b) A low to high transition on the Data Carrier Detect signal line

"0" - Cleared by selecting the Receiver Data Register or by resetting the Receiver Interrupt Enable Bit.

CLOCK INPUTS

Separate high impedance TTL compatible inputs are provided for clocking of transmitted and received data. Clock frequencies of 1, 16, or 64 times the data rate may be selected.

A. Transmit Clock (TXC)

The transmit clock input is used for the clocking of transmitted data. The transmitter initiates data on the negative transition of the clock.

B. Receive Clock (RXC)

The Receive Clock input is used for synchronization of received data. The receiver strokes the data on the positive transition of the clock. (In the $\div 1$ mode, the clock and data must be synchronized externally).

MODEM CONTROL

The ACIA includes several functions that permit limited control of a data modem. The functions included are Clear-to-Send, Request-to-Send and Data Carrier Detect.

A. Clear-to-Send ($\overline{\text{CTS}}$)

This high impedance TTL compatible input provides automatic control of the transmitting end of a communications link via the modem's "clear-to-send" active low output.

B. Request-to-Send ($\overline{\text{RTS}}$)

The Request-to-Send output enables the MPU to control a modem via the data bus. The active state is low.

C. Data Carrier Detected ($\overline{\text{DCD}}$)

This high impedance TTL compatible input provides automatic control of the receiving end of a communication link by means of the modem "Data-Carrier-Detect" or "Received-Line-Signal Detect" output. The $\overline{\text{DCD}}$ input inhibits and initializes the receiver section of the ACIA when high. A low to high transition of the Data Carrier Detect initiates an interrupt to the MPU to indicate the occurrence of a loss of carrier.

RECEIVED DATA LINE (RX)

The Received Data line is a high impedance TTL compatible input through which data is received in a serial NRZ (Non Return to Zero) format. Synchronization with a clock for detection of data is accomplished internally when clock rates of 16 or 64 times the bit rate are used. Data rates are in the range of 0 to 500Kbps when external synchronization is utilized.

TRANSMITTED DATA LINES (TX)

The Transmit Data Output line transfers serial NRZ data to a modem or other peripheral at the same range of rates as the received data.

$\div 16 = 115.2 \text{ kHz}$